Self-adaptive Service Organization for Pragmatics-aware Service Discovery

Dionysis Athanasopoulos
School of Engineering & Computer Science
Victoria University of Wellington, New Zealand
dionysis.athanasopoulos@ecs.vuw.ac.nz

***Abstract*—Traditional service registries form groups of functionally related services based on service descriptions, without taking into account past service-usage from the consumers' perspective, a.k.a. pragmatics. Thus, we propose a self-adaptive service-organization mechanism that follows an iterative and evolutionary life-cycle for autonomically evolving service groups by the arrival of pragmatics. Since pragmatics are not available beforehand, we propose an on-line service-organization algorithm that leverages the highly accumulated number of pragmatics. We evaluate the efficiency and effectiveness of our mechanism on the services of a publicly available benchmark and the results show that the effectiveness of the traditional service-organization is improved, while a low number of pragmatics is greedily stored.**

***Keywords*-service organization; self-adaptive; on-line hierarchical clustering; pragmatics.**

I. Introduction

Whereas famous vendors (e.g. `Google`, `Amazon`) provide access to their resources via adopting the Service-oriented Architecture (SoA) style, the majority of the available Web services has not been discovered or invoked [1]. The process for discovering available services typically includes the submission of service requests to registries, which return a set of service candidates by using matching mechanisms. Registries form organization schemes that are consisted of groups of functionally similar services[1]. Services' functionality is specified based on *the providers' perspective* using syntactic (e.g. WSDL) descriptions, usually enriched with semantics (e.g. WSDL-S, OWL-S). Registries may further form classification schemes, consisted of classes of groups. Each class is usually characterized by a label [2]. However, *traditional registries do not organize/classify services considering past service usage from the consumers' perspective*, a.k.a. *pragmatics* [3].

Pragmatics are either explicit, i.e. consumers specify them (e.g. XML-based queries [4]), or implicit, i.e. past service usage. Since explicit pragmatics requires an incentive for consumers to provide them, we consider only the case of implicit pragmatics. While the importance of pragmatics has been underlined [5], a few early approaches[2] have been proposed that offer pragmatics-aware searching facilities [4–7], but not service organization/classification. The restriction of the approaches is that they compare pragmatics-aware queries to traditional service descriptions. To overcome this restriction, *service-organization mechanisms should dynamically enhance service descriptions with pragmatics and accordingly evolve organization schemes*.

Contribution. We propose a mechanism that organizes services into groups based on both service interfaces and implicit pragmatics. As an indicative aspect of service pragmatics, we consider previously used schema instances, given as input to operation invocations[3]. To be responsive in dynamically arrived pragmatics, the mechanism enters into an iterative and evolutionary life-cycle, during which it autonomically evolves its organization scheme. In particular, we propose a *self-adaptive* mechanism [8] that goes through an off-line phase to produce an initial organization scheme and an on-line phase to evolve the scheme based on pragmatics.

During the on-line phase, our mechanism initially enhances service interfaces with newly arrived schema instances. To do so, it uses an extended conceptual model for service-interfaces that permits such an enhancement. The mechanism also re-calculates the similarities between the organized services via using our semantic edit-distance metric for comparing schema instances.

since schema instances are not available beforehand, our mechanism adopts an *on-line* hierarchical clustering algorithm to evolve organization schemes. Due to the highly accumulated number of instances, our algorithm faces a time and space efficiency challenge. The recent space-efficient clustering algorithm in [9] stores all of the newly-arrived objects and apply compression techniques for storing their similarities. To a different direction, we extend the classical on-line clustering algorithm in [10] by proposing a greedy algorithm that stores the top-k instances of each service and permanently discards the remaining instances.

To evaluate the effectiveness and efficiency of our approach, we implement the mechanisms and metrics, used by the on- and off-line phases. The results show the effectiveness of the traditional service-organization is improved, while a low number of pragmatics is greedily stored.

Our contribution is summarized and structured in Sections as follows. Section II categorizes and compares related approaches. Section III specifies our pragmatics-aware

[1]The non-functional service organization is not covered in this work.

[2]Our work is not related to service-recommendation approaches that consider historical usage data for predicting consumers' interests.

[3]More aspects of pragmatics will be used in the next mechanism version.

conceptual model of service interface. Section IV defines our pragmatics-aware service-similarity metric. Section V describes the proposed life-cycle of self-adaptive service-organization. Section VI specified the modus operandi of the on-line service-organization algorithm. Section VII presents the experimental evaluation. Section VIII summarizes our approach and discusses its future research directions.

II. Related Work

Our work belongs to the fields of the service discovery (Section II-A) and self-adaptive software (Section II-B).

A. Service Discovery

We categorize the approaches along two dimensions: service organization and pragmatics-aware service discovery. Due to the vast number of approaches in the first dimension, we describe the top approaches based on the ranking in [11].

Service organization. It is performed in a *top-down* (schemes are pre-defined and incrementally populated) or *bottom-up* fashion (groups/classes are reverse-engineered).

1) Top-down: The majority of the approaches uses centralized UDDI[4] registries. [12] extends UDDI's keywords-based searching-facilities with ontology-based matching. There are also distributed approaches, which use Peer-to-Peer (P2P) nodes for indexing the service corpus. The underlying registry of a peer can be a UDDI registry [13], [14] or a database system [15]. [2], [16], [17] do not use UDDI or P2P nodes, but they define service (XML-based [16], label-based [2], OWL-S–based [17]) abstractions as representatives of service groups/classes.

2) Bottom-up: The majority of the approaches applies clustering techniques. [18] and [20] clusters similar terms and tags, respectively, contained in non-semantic services. [19] clusters non-semantic services and extracts service abstractions as cluster representatives. [22] presents an ant-inspired method for clustering semantic services. [21] proposes clustering for classifying semantic services. [23] clusters RESTful services. [24] clusters services through mining semantic information from service interaction.

Comparing the approaches in terms of the used service descriptions, organization process and fashion (Table I(a)), we observe *our approach is the only (i) self-adaptive and (ii) (bottom-up) on-line approach that (iii) considers pragmatics*.

Pragmatics-aware service discovery. One of the earliest approaches [5] selects services based on collaborative lattices, built by using agents for collecting ratings about service providers. [6] retrieves services based on inferring rules, extracted from the names of previously used service operations (i.e. high-level pragmatics). [4] retrieves services considering explicit pragmatics in which consumers specify the reason and context of service requests. A very recent approach [7] selects services considering previous developers' selection choices, based on their social networks.

[4] uddi.xml.org

Table I
Summary and comparison of service-discovery approaches.

(a) Service organization.

	Service Description	Organization		
		process	fashion	scheme
[2]	WSDL	traditional	top-down	classes
[12]	OWL-S			P2P & UDDI
[13] [14]	WSDL-S			
[15]				P2P & DB
[17]	OWL-S			abstractions
[16]	WSDL			
[18]	WSDL	traditional	bottom-up (off-line)	clusters
[19]				abstractions
[20]				clusters
[21] [22]	OWL-S			
[23]	REST			
[24]	WSDL			
Ours	*pragmatics*	*self-adaptive*	*on-line*	abstractions

(b) Pragmatics-aware service discovery.

	Pragmatics	Technique	Facility
[4]	explicit queries	matching	retrieval
[5]	providers' rating	concept-lattice	selection
[6]	operation names	inferring rules	retrieval
[7]	developers' choices	ranking	selection
Ours	*schema instances*	*Semantic edit distance*	*organization*

Comparing the approaches in terms of the used pragmatics, underlying technique, and offered facility (Table I(b)), *our approach is the only (i) service-organization approach that (ii) considers low-level pragmatics* (schema instances).

B. Self-adaptive Software

Self-adaptive software autonomically evolves in response to changes, caused by external sources [8]. Self-adaptive SoA approaches do not focus on the service-organization task. In particular, [25] focuses on integration problems that derive from changes in used services and [26] proposes a formal model for the actions taken by self-adaptive SoA software. To the best of our knowledge, *our approach is the first self-adaptive service-organization approach*.

III. Pragmatics-aware Service-Interface Model

Our conceptual model of service interface is derived by the WSDL-based specification of services and extended with data-structures suitable for storing the top-k schema instances of each service interface. In particular, a service interface `SI` (Table II (Eq. 1)) is characterized by its `name` and set of operations `OPs` (Table II (Eq. 2)). An operation `OP` (Table II (Eq. 3)) accepts an input message `in` and produces an output message `out`. A message `M` (Table II (Eq. 4)) is characterized by its `id` (it is unique across all of the available services), `name`, XML schema `S`, and top-k schema instances. In each message, only the instances `d[k]` of its schema are stored that are the most similar to

Table II
THE DEFINITION OF THE PRAGMATICS-AWARE SERVICE MODEL.

$$SI := (name : \texttt{String},\ ops : OPs) \quad (1)$$
$$OPs := \{op_i : OP\} \quad (2)$$
$$OP := (name : \texttt{String},\ in : M,\ out : M) \quad (3)$$
$$M := \Big(id : \texttt{int}, name : \texttt{String},\ s : S,\ d[k] : D[k], sim_D[k] \in [0,\ 1]^k, id[k] : \texttt{int}[k]\Big) \mid k \in \mathbb{N} \quad (4)$$
$$S := e : E \quad (5)$$
$$E := \Big(name : \texttt{String},\ type : \texttt{anyType},\ \{e_i : E\}\Big) \quad (6)$$
$$D := l : L \mid D \text{ is an instance of } S \quad (7)$$
$$L := \Big(name : \texttt{String},\ value : \texttt{String},\ \{l_i : L\}\Big) \quad (8)$$

instances of other service messages `id[k]`. Their similarity values sim_D`[k]` are also stored. The arrays `d`, sim_D, and `id` (hereafter called *pragmatics arrays*) have position correspondence (Table II (Eq. 4)).

Message schema is represented as rooted tree of elements (Table II (Eq. 5)). Each element is characterized by its `name`, built-in `type` (subtype of XML `anyType`[5]), and children (Table II (Eq. 6)). We consider only the name and built-in type attributes of element, since they play the most important role in element similarity [27]. Similarly, an instance (XML document) `D` of schema, it is also represented as rooted tree of labels (Table II (Eq. 7)). Each label `L` is characterized by its `name`, `value`, and children (Table II (Eq. 8)).

Illustrative example. We consider three services that manage information about pre-, post-graduate and PhD students. Fig. 1 represents the example service-interfaces by using our model. It also represents two example schema-instances[6], along with the content of the pragmatics arrays.

IV. PRAGMATICS-AWARE SERVICE SIMILARITY

To calculate the service-interface similarity, we follow the hierarchical structure of the service-interface model and further consider the stored schema-instances, as follows.

A. Suite of Similarity Metrics for Service Interfaces

Interface similarity. The similarity metric sim_{SI} (Table III (Eq. 1)) for two service interfaces is defined as the arithmetic mean of (i) the complement of the normalized edit distance `NED` (Table III (Eq. 2 and 3)) between the interface names[7] and (ii) the arithmetic mean of the similarities of their matched operations. To find the best possible operation matching M_{OP} (Table III (Eq. 4)), we solve the optimization problem of the maximum weighted matching in a bipartite graph [29]. The nodes of the graph correspond to the operations of the interfaces, while the edges correspond to the similarities of the operation pairs.

Operation similarity. The similarity metric sim_{OP} (Table III (Eq. 5)) for two service operations is defined as the arithmetic mean of (i) the normalized edit distance between the operation names and (ii) the arithmetic mean of the similarities of the input and output operation messages.

Message similarity. The similarity metric sim_M (Table III (Eq. 6)) for two messages is defined as the arithmetic mean of (i) the normalized edit distance between the message names and (ii) the aggregation of their schema and instance similarities (bold font in Table III). We assume that instance similarity has higher priority than schema similarity. To this end, we propose the priority-based aggregation function $\mathcal{F}_{PR}$ (Table III (Eq. 13)), in which the prioritized role of the `x` objective is reflected by the fact that `x` contributes with its entire value. On the contrary, the non-prioritized `y` objective contributes with its product with `x`, which is of lower magnitude order than the `x` value (`x`, `y` $\in [0, 1]$). Regarding the similarity of schema instances, motivated by the fact that similar instances are usually specified by similar hierarchical structures [30], we propose in Section IV-B a semantic metric that compares tree structures.

Schema similarity. The similarity metric sim_S (Table III (Eq. 7)) for two schemas is defined as the arithmetic mean of the similarities of the matched schema elements. To find the best possible schema matching M_S (Table III (Eq. 9))), we solve again the maximum weighted matching problem.

Element similarity. The similarity metric sim_E (Table III (Eq. 8)) for two elements is defined as the arithmetic mean of (i) the normalized edit distance between the element names and (ii) the similarity between the element built-in types (sim_T metric). sim_T values are statically defined in [27].

B. Semantic Edit-distance Metric for Schema Instances

The recent robust tree edit-distance (RTED) technique [31] converts the compared trees into strings (their format is depicted at the bottom of Fig. 1). Following, it counts the min number of the operations (relabel, insert, delete) required to transform one tree into another, without though dealing with semantic similarities (i.e. exact label matching). The RTED result is the optimal edit script (i.e. the sequence of operations) for the compared trees. Without changing the algorithmic logic of RTED, our metric takes as input a RTED edit-script `ES` (Table III (Eq. 11)), which is defined in [31], and computes a new edit-distance value as follows. For each pair of matched labels, if they are internal nodes, the metric calculates the normalized edit distance (Levenshtein's metric [28]) of their names (first branch in Table III (Eq. 12)). If the matched labels are tree leaves (a.k.a. real data), the metric considers only the case of numeric data, calculating their absolute difference divided by the absolute value of the min numeric datum (second branch in Table III (Eq. 12))[8]. Note

[5] www.w3.org/TR/xmlschema-2
[6] Due to lack of space, we present instances for the first two services.
[7] We adopt the Levenshtein's string-based metric [28].
[8] We leave as future work the comparison of other kinds of data, since they depend on the service domain (domain-specific ontology is needed).

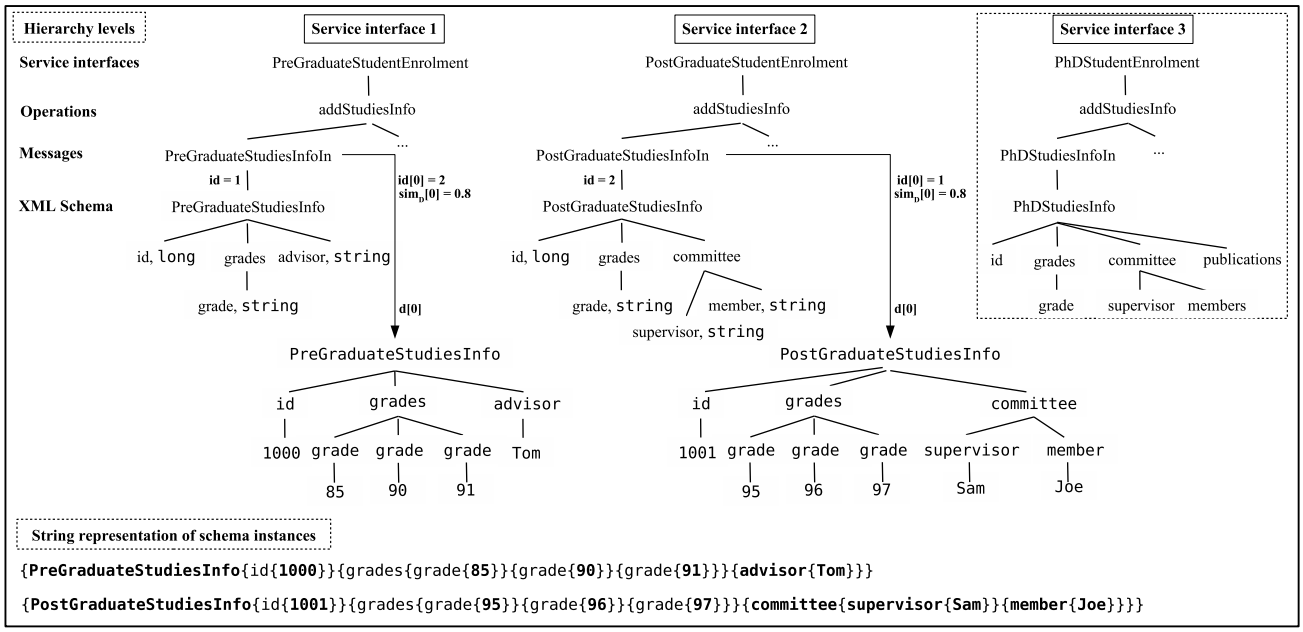

Figure 1. The representation of the example service-interfaces by using the pragmatics-aware service-interface model.

our metric is communitative, since RTED and Levenshtein's metrics are communitative. Overall, the similarity between two schema instances sim_D (Table III (Eq. 10)) equals to the complement of the normalized[9] `TED`.

Illustrative example. Returning to the previous example (Fig. 1), the RTED similarity of the schema instances equals to 10 (normalized value, 0.63) and is identified by the different label pairs (bold font). On the contrary, their sim_D similarity equals to 5.4 (normalized value, 0.8).

V. Life-cycle of Self-adaptive Organization

The proposed life-cycle is in accordance to the generic one for self-adaptive software [8], which includes the off-line and on-line phases. In our case, the off-line phase corresponds to the `Initial Service Organization` (Section V-A), while the on-line phase to the `Self-adaptive Service Organization` (Section V-B).

A. Initial Service Organization

This phase is depicted at the left-hand side of Fig. 2. Since we focus on the second phase, we assume that the first phase adapts an existing off-line hierarchical clustering mechanism [32]. The mechanism is adapted to use our service-interface model and similarity metrics.

Illustrative example. Giving as input the three service-interfaces of Fig. 1 to the off-line clustering mechanism, the dendrogram of clusters at the left-hand side of Fig. 4 is produced. The off-line clustering method at its first iteration merges (si_1, si_2), since their similarity is greater than the similarities of the remaining pairs, (si_1, si_3) and (si_2, si_3).

[9]We normalize by dividing with the sum of the sizes of the two instances.

B. Self-adaptive Service Organization

The second phase is depicted at the right-hand side of Fig. 2. During this phase, the `On-line Calculation of Instance Similarity` and `On-line Evolution of Organization Scheme` mechanisms are executed, whose algorithmic details are provided in Section VI.

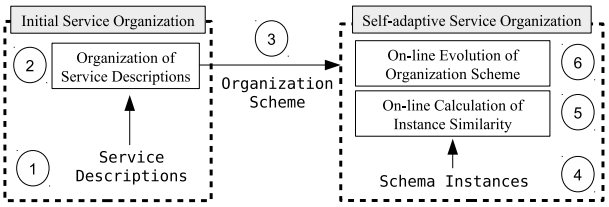

Figure 2. The life-cycle of the self-adaptive service organization.

VI. Self-adaptive Service Organization

The `On-line Calculation of Instance Similarity` is iteratively executed by the arrival of new schema instances. If the content of the pragmatics arrays is not updated for a fixed continuous number of newly arrived instances, then the `On-line Evolution of Organization Scheme` is executed for releasing a new version of the scheme. The underlying algorithms and their complexity, are detailed in Sections VI-A and VI-B.

A. On-line Calculation of Instance Similarity

Algorithm 1 accepts a schema instance d_x, the constant `k`, and the list of the service interfaces `sis` that are contained in the organization scheme (input line in Alg. 1). The algorithm

Table III
THE DEFINITION OF THE PROPOSED METRIC FOR CALCULATING SERVICE-INTERFACE SIMILARITY.

$$sim_{SI}\Big(si_1 : S,\ si_2 : S\Big) := \frac{\Big(1 - NED(si_1.name,\ si_2.name)\Big) + \frac{\sum_{\forall (op_i,\ op_j)\ \in\ M_{OP}} sim_{OP}(op_i,\ op_j)}{|M_{OP}(si_1,\ si_2)|}}{2} \tag{1}$$

$$NED\Big(n_1 : \texttt{String}, n_2 : \texttt{String}\Big) := \frac{2 * ED(n_1, n_2)}{|n_1|+|n_2|+ED(n_1, n_2)} \ \Big|\ |n_1| \text{ and } |n_2| \text{ are the lengths of } n_1 \text{ and } n_2, \text{ respectively.} \tag{2}$$

$$ED\Big(n_1 : \texttt{String}, n_2 : \texttt{String}\Big) := |n_1|+|n_2|-2 * lcs(n_1, n_2) \ |\ lcs \text{ is the length of their longest common substring.} \tag{3}$$

$$M_{OP}\Big(si_1 : SI,\ si_2 : SI\Big) := \Big\{(op_i,\ op_j) \in si_1.OPs \times si_2.OPs\Big\} : \sum_{\forall (op_i,\ op_j)} sim_{OP}(op_i,\ op_j) \text{ is minimized.} \tag{4}$$

$$sim_{OP}\Big(op_1 : OP,\ op_2 : OP\Big) := \frac{\Big(1 - NED(op_1.name,\ op_2.name)\Big) + \frac{sim_M(op_1.in,\ op_2.in)+sim_M(op_1.out,\ op_2.out)}{2}}{2} \tag{5}$$

$$sim_M\Big(m_1 : M, m_2 : M\Big) := \frac{\Big(1 - NED(m_1.name, m_2.name)\Big) + \mathbf{F_{PR}}\Big(\mathbf{m_1.sim_D[i]},\ \mathbf{sim_S(m_1.s, m_2.s)}\Big)}{2} \ \Big|\ m_1.id[i] = m_2.id \tag{6}$$

$$sim_S\Big(s_1 : S, s_2 : S\Big) := \frac{\sum_{\forall (s_1.e.e_i,\ s_2.e.e_j)\ \in\ M_S} sim_E(s_1.e.e_i,\ s_2.e.e_j)}{|M_S(s_1,\ s_2)|} \tag{7}$$

$$sim_E\Big(e_1 : E, e_2 : E\Big) := \frac{\Big(1 - NED(e_1.name,\ e_2.name)\Big) + sim_T(e_1.type, e_2.type)}{2} \tag{8}$$

$$M_S\Big(s_1 : S,\ s_2 : S\Big) := \Big\{(e_i,\ e_j) \in s_1 \times s_2\Big\} : \sum_{\forall (e_i,\ e_j)} sim_E(e_i,\ e_j) \text{ is minimized.} \tag{9}$$

$$sim_D\Big(d_1 : D,\ d_2 : D\Big) := 1 - \frac{TED(d_1,\ d_2)}{|d_1|+|d_2|} \ \Big|\ |d_1| \text{ and } |d_2| \text{ are the sizes (i.e. number of labels) of } d_1 \text{ and } d_2, \text{ respectively.} \tag{10}$$

$$TED\Big(d_1 : D,\ d_2 : D\Big) := \sum_{\forall (l_i,\ l_j)\ \in\ ES(d_1,d_2)} cost(l_i,\ l_j) \tag{11}$$

$$cost\Big(l_1 : L, l_2 : L\Big) := \begin{cases} NED(l_1.name,\ l_2.name) & \text{if } l.\{l_i\} = \emptyset \text{ (internal node)} \\ \frac{|l_1.value - l_2.value|}{|MIN(l_1.value, l_2.value)|} & \text{if } l_1.value, l_2.value \in \mathbb{R} \\ 1 & \text{otherwise} \end{cases} \tag{12}$$

$$\mathcal{F}_{PR}\Big(x \in [0,\ 1], y \in [0,\ 1]\Big) = \frac{x + x * y}{2} \tag{13}$$

returns the same list of service-interfaces `sis` (output line in Alg. 1) with potentially updated their pragmatics arrays.

Technically, the algorithm compares d_x against the stored instances of the other service interfaces (Alg. 1 (1-6)) and by using our sim_D metric, it identifies the instance with the maximum similarity (Alg. 1 (7-11)). The algorithm compares instances only if their schemas are similar (Alg. 1 (6)). Following, the algorithm checks if the max similarity is greater than at least one of the top-k similarities that have been stored in the corresponding message of d_x (Alg. 1 (12-14)). If it holds, then the algorithm updates the pragmatics arrays of this message (Alg. 1 (15-19)).

Each message stores its own list of top-k instances, since this list is not necessarily the same with those of other messages. However, if the same pair of instances appears in two lists, then their similarities are very close to each other or identical, as shown in the evaluation results.

Complexity. For a new instance, the algorithm iterates over the existing interfaces, their operations, and the stored instances. Thus, the time and space complexity of the algorithm is captured by the linear expression, $\mathcal{O}(\texttt{k*|sis|*|ops|})$. Note that the above product equals to the total number of the stored schema instances in all service interfaces.

Illustrative example. Returning to our running example, the mechanism is iteratively executed giving as input the three services, randomly generated schema instances[10], and the value 2 for `k`. The mechanism execution stops when no changes are performed in the pragmatics arrays, whose final content is depicted in Fig. 3(a). We observe that the instances of the pair (si_2, si_3) have the highest similarity, in contrast to the pair (si_1, si_2) identified by the off-line mechanism. We also observe that there is a convergence in the similarities of the bidirectional pairs.

	si_1		si_2		si_3	
$\mathbf{sim_D}$	0.59	0.53	0.62	0.57	0.63	0.53
id	2	3	3	1	2	1
d	118	159	133	110	81	112

(a) Schema instances.

	$\mathbf{sim_{SI}}$		$\mathbf{sim_{SI}}$
(si_1, si_2)	0.56	(si_2, si_1)	0.48
(si_2, si_3)	0.55	(si_1, si_3)	0.43
(si_1, si_2)	0.51	(si_3, si_1)	0.42

(b) Service interfaces.

Figure 3. The results of our mechanisms for the running example.

[10]The generator of schema instances is described in Section VII.

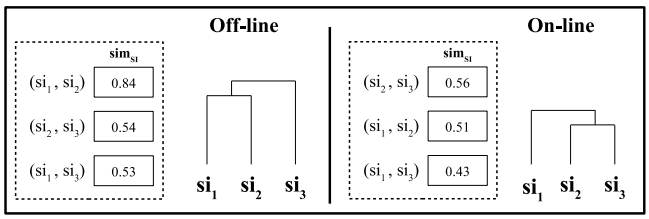

Figure 4. The dendrograms produced with clustering for the example.

B. On-line Evolution of Organization Scheme

The mechanism calculates the similarities of the service-interface pairs, whose schema instances have been stored in the pragmatics arrays, and stores the pairs sorted with respect to their similarities. Then, it starts from the pair that has the highest similarity and clusters service interfaces contained in the same pair, until no more clusters can be formed (i.e. their similarities equal to zero) or only one cluster remains. Since the modus operandi of the mechanism is straightforward, we do not specify it in an algorithmic format.

Complexity. Since the mechanism iterates over the existing interfaces, their operations, and the stored instances, its time and space complexity scales with the same linear expression with the previous mechanism. This expression does not scale with the quadratic number of all possible interface pairs as happens in the classical clustering.

Illustrative example. Returning to our running example, the second mechanism is executed for the results of the first mechanism. From the new results (Fig. 3(b)), we observe that the similarities of bidirectional pairs converge to close values and that the pair, (si_2, si_3), has the highest similarity. Consequently, the produced dendrogram (Fig. 4) differs from that of the off-line mechanism.

VII. Experimental Evaluation

We implemented the mechanisms and metrics, used by the on-line and off-line phases. The executable file of our research prototype is available online at this location[11]. We evaluate the effectiveness and efficiency of the mechanisms on the Web services of the benchmark, `OWLS-TC4`[12]. We firstly set up our experiments.

A. Experiment Setup

Benchmark. `OWLS-TC4` has been used by many state-of-the-art approaches that calculate service similarity (e.g. [19], [27]). The majority of the `OWLS-TC4` services was retrieved from public IBM UDDI registries. Each service document contains only one interface, which provides a single operation. `OWLS-TC4` also provides the result sets of 42 queries, Q_1-Q_{42}, which correspond to groups of semantically similar services. Due to lack of space, we present the evaluation results for Q_1-Q_{20}, but we reached to analogous conclusions for the remaining queries. Finally, `OWLS-TC4` provides ideal matchings between the schema elements of the services, based on the ontology classes to which the elements belong.

Algorithm 1 On-line Calculation of Instance Similarity

Input: $d_x : D,\ k : \texttt{int},\ sis = \{si_1 : SI, si_2 : SI, \dots\}$
Output: $sis = \{si_1 : SI, si_2 : SI, \dots\}$

```
1: si_x ← IDENTIFY(d_x) | d_x is instance of si_x.ops.op.in.s
2: for all si_i ∈ sis do
3:     if si_x == si_i then continue end if
4:     for all op_j ∈ si_i.ops do
5:         s_j ← op_j.in.s
6:         if sim(s_x, s_j) > 0 then
7:             Double max ← −1
8:             for all d[z_1] ∈ op_j.in do
9:                 Double sim ← sim_D(d_x, d[z_1])
10:                if sim > max then max ← sim end if
11:            end for
12:            for all d[z_2] ∈ op_x.in do
13:                if max < d[z_2] then break end if
14:            end for
15:            if z_2 <= k then
16:                in_x.d[z_2] ← d_x
17:                in_x.sim_D[z_2] ← max
18:                in_x.id[z_2] ← in_j.id
19:            end if
20:        end if
21:    end for
22: end for
```

Methodology. We firstly executed the off-line phase and then, the on-line phase. The mechanism, `On-line Calculation of Instance Similarity`, is iteratively executed for randomly generated schema instances. If none pragmatics-array is updated for 10 continuous iterations, the mechanism, `On-line Evolution of Organization Scheme`, is executed. Both mechanisms are executed for the value 2 of `k` as a representative low `k` value, since high values negatively affect the mechanism efficiency. Thus, our evaluation includes two parts, one part for each mechanism (Sections VII-B and VII-C).

We also implemented a generator of XML documents. It creates for each schema element a label based on its ontology class. If two labels are numeric, then the generator produces close values. The number of the label instances is randomly selected and belongs to the interval of the min and max occurrence numbers of the corresponding element.

B. Evaluation of On-line Calculation of Instance Similarity

For each `OWLS-TC4` query, we compare the instance similarities against the similarities of the corresponding schemas in order to examine if their values are close (their

[11] ecs.victoria.ac.nz/foswiki/pub/Main/DionysisAthanasopoulos/pragmatics.zip
[12] projects.semwebcentral.org/projects/owls-tc
[13] We do not include Q_2, since its result set contains one service interface.

Table IV
EVALUATION RESULTS OF THE ON-LINE SERVICE ORGANIZATION FOR THE SERVICE INTERFACES OF THE OWLS-TC4 QUERIES.

Queries[13]		On-line Calculation of Instance Similarity				On-line Evolution of Organization Scheme	
		#stored / #all pairs	#bidirectional / #different pairs	#bidirectional (close values)	schema vs. instance (#pairs - close values)	on-line vs. off-line ranked pairs: #common / #different	on-line vs. off-line ranked pairs: #repositions / #common
Q_1	**L**	22 / 110 (20%)	9 / 13 (69%)	9 (100%)	18 (82%)	8 (57%)	5 (63%)
Q_3	**S**	10 / 20 (50%)	4 / 6 (67%)	3 (75%)	9 (90%)	6 (100%)	0 (0%)
Q_4	**L**	22 / 110 (20%)	7 / 15 (47%)	7 (100%)	7 (32%)	12 (71%)	10 (83%)
Q_5	**L**	28 / 182 (15%)	6 / 22 (27%)	6 (100%)	21 (75%)	14 (64%)	13 (93%)
Q_6	**L**	16 / 56 (29%)	4 / 12 (33%)	4 (100%)	15 (94%)	8 (62%)	15 (100%)
Q_7	**L**	42 / 420 (10%)	14 / 28 (50%)	13 (93%)	22 (52%)	9 (33%)	9 (100%)
Q_8	**S**	10 / 20 (50%)	3 / 7 (43%)	3 (100%)	8 (80%)	6 (86%)	8 (83%)
Q_9	**S**	6 / 6 (100%)	3 / 3 (100%)	3 (100%)	6 (100%)	3 (100%)	3 (100%)
Q_{10}	**S**	6 / 12 (50%)	3 / 3 (100%)	3 (100%)	3 (50%)	3 (100%)	2 (67%)
Q_{11}	**S**	6 / 6 (100%)	3 / 3 (100%)	3 (100%)	2 (33%)	3 (100%)	0 (0%)
Q_{12}	**L**	16 / 56 (29%)	4 / 12 (33%)	4 (100%)	14 (88%)	9 (75%)	8 (89%)
Q_{13}	**L**	28 / 182 (15%)	8 / 20 (40%)	8 (100%)	25 (89%)	11 (55%)	25 (100%)
Q_{14}	**S**	10 / 20 (50%)	4 / 6 (67%)	4 (100%)	5 (50%)	4 (67%)	4 (100%)
Q_{15}	**S**	10 / 20 (50%)	3 / 7 (43%)	3 (100%)	4 (40%)	5 (71%)	5 (100%)
Q_{16}	**S**	8 / 12 (67%)	3 / 5 (60%)	3 (100%)	8 (100%)	8 (80%)	1 (25%)
Q_{17}	**L**	42 / 420 (10%)	11 / 31 (35%)	11 (100%)	30 (71%)	16 (52%)	14 (88%)
Q_{18}	**L**	16 / 56 (29%)	6 / 10 (60%)	6 (100%)	8 (50%)	8 (80%)	8 (100%)
Q_{19}	**S**	8 / 12 (67%)	3 / 5 (60%)	3 (100%)	4 (50%)	4 (80%)	3 (75%)
Q_{20}	**L**	36 / 306 (12%)	7 / 29 (24%)	7 (100%)	21 (58%)	9 (31%)	9 (100%)
Small-sized		65%	71%	97%	71%	87%	61%
Large-sized		19%	42%	99%	69%	58%	92%

difference ≤ 0.2) and if the similarities of the bidirectional pairs converge to close values (their difference $\leq$ 0.1). Moreover, we counted the number of the stored schema instances, since the mechanism efficiency depends on it.

The results are depicted in Table IV (from the second to fifth column). At the bottom of Table IV, we also calculated the average results for large- (annotated by **L**) and small-sized queries (**S**, |pairs| ≤ 20). As expected, the percentage of the stored pairs is low, 19%, for large-sized queries and medium, 65%, for small-sized queries. The percentage of the bidirectional pairs is medium, 42%, for large-sized queries and medium-high, 71%, for small-sized queries. This result verifies our previously mentioned intuition that the lists of the top-k instances are not the same across the services. The percentage of the bidirectional pairs that converges to close values is very high, 99% and 97%, for large- and small-sized queries, respectively. Finally, the medium-high percentage of the 69% and 71% for large- and small-sized queries, respectively, has instance similarity close to schema similarity. We examine below the impact of the complements of these percentages on the service organization.

C. Evaluation of On-line Evolution of Organization Scheme

For each `OWLS-TC4` query, we compare the ranked sets of service-interface pairs of the two phases. In particular, we compare the percentages of their common pairs and repositions. From the results (6th and 7th columns of Table IV), the percentage of the common pairs (resp. repositions) is high, 87% (resp. medium, 61%), for small-sized queries, and medium, 58% (resp. high, 92%), for large-sized queries. To explain the results, we calculated the effectiveness of the off-line service-similarity and we inspected the ranked sets. We calculated the effectiveness of service similarity against the ideal matchings via using the *F-measure* metric [33].

We present in Fig. 5 a part of our inspection results for two representative queries (Q_9 and Q_4, small- and large-sized, respectively). Since the off-line similarity and F-measure values equal to 1.0 in Q_9, very small differences in the on-line similarities led to repositions. In Q_4, the differences between the instance and schema similarities (see previous subsection), along with the usage of the constraint `k`, led to discard the pairs, (si_4, si_3) and (si_3, si_2). However, these repositions improved the organization results, since the pair (si_4, si_2) correctly[14] is of much higher similarity than those of the other pairs. Concluding, instances helped in distinguishing similar pairs in a fine-grained way.

VIII. CONCLUSIONS & FUTURE WORK

We proposed a self-adaptive service-organization mechanism that follows an iterative life-cycle for autonomically evolving organization schemes by the arrival of pragmatics. Since pragmatics are not available beforehand, we proposed an on-line algorithm. We evaluated the efficiency and effectiveness of our mechanism on the services of a publicly available benchmark and the results show the effectiveness of the traditional service-organization is improved, while a low number of pragmatics is greedily stored.

Our future research purpose is to extend our service-interface model and on-line algorithm to use other aspects of service pragmatics (e.g. service-usage scenarios). An interesting direction is to extend our approach to cope with big real-time monitoring data. A final direction is to propose a user-collaborative platform that gathers service pragmatics

[14] si_4 = `monograph`, si_3 = `book_price`, si_2 = `printed_material`.

	Off-line		On-line
	sim_{SI}	F-measure	sim_{SI}
(si_1, si_2)	1.00	1.00	0.99
(si_2, si_3)	1.00	1.00	0.98
(si_1, si_3)	1.00	1.00	0.99

(a) Small-sized query.

Q_4	Off-line		On-line
	sim_{SI}	F-measure	sim_{SI}
(si_4, si_3)	0.78	0.78	[illegible]
(si_4, si_2)	0.78	0.78	0.85
(si_3, si_2)	0.78	0.78	[illegible]

(b) Large-sized query.

Figure 5. Detailed evaluation results for two OWLS-TC4 queries.

REFERENCES

[1] W. Chen, I. Paik, and P. C. K. Hung, "Constructing a global social service network for better quality of web service discovery," *IEEE Transactions on Services Computing*, vol. 8, no. 2, pp. 284–298, 2015.

[2] X. Liu, S. Agarwal, C. Ding, and Q. Yu, "An LDA-SVM active learning framework for web service classification," in *International Conference on Web Services*, 2016, pp. 49–56.

[3] W. Rong and K. Liu, "A survey of context aware web service discovery: From user's perspective," in *International Symposium on Service-Oriented System Engineering*, 2010, pp. 15–22.

[4] E. Tamani and P. Evripidou, "A pragmatic methodology to web service discovery," in *International Conference on Web Services*, 2007, pp. 1168–1171.

[5] R. M. Sreenath and M. P. Singh, "Agent-based service selection," *Journal of Web Semantics*, vol. 1, no. 3, pp. 261–279, 2004.

[6] N. Kokash, A. Birukou, and V. D'Andrea, *Web Service Discovery Based on Past User Experience*, 2007.

[7] D. Bianchini, V. D. Antonellis, and M. Melchiori, "An approach for service selection based on developers' ranking," in *International Conference on Web Services*, 2016, pp. 704–707.

[8] J. Andersson, L. Baresi, N. Bencomo, R. de Lemos, A. Gorla, P. Inverardi, and T. Vogel, "Software engineering processes for self-adaptive systems," in *Software Engineering for Self-Adaptive Systems II - International Seminar*, 2010, pp. 51–75.

[9] T. Nguyen, B. Schmidt, and C. K. Kwoh, "Sparsehc: A memory-efficient online hierarchical clustering algorithm," in *International Conference on Computational Science*, 2014, pp. 8–19.

[10] Y. El-Sonbaty and M. A. Ismail, "On-line hierarchical clustering," *Pattern Recognition Letters*, vol. 19, no. 14, pp. 1285–1291, 1998.

[11] M. Rambold, H. Kasinger, F. Lautenbacher, and B. Bauer, "Towards autonomic service discovery - a survey and comparison," in *International Conference on Services Computing*, 2009, pp. 192–201.

[12] Q. Qiu, Q. Xiong, Y. Yang, and F. Luo, "Study on ontology-based web services discovery," in *International Conference on Computer Supported Cooperative Work in Design*, 2007, pp. 641–645.

[13] K. Verma, K. Sivashanmugam, A. P. Sheth, A. A. Patil, S. A. Oundhakar, and J. A. Miller, "METEOR-S WSDI: A scalable P2P infrastructure of registries for semantic publication and discovery of web services," *Information Technology and Management*, vol. 6, no. 1, pp. 17–39, 2005.

[14] M. Pantazoglou, A. Tsalgatidou, and G. Athanasopoulos, "Discovering web services and JXTA peer-to-peer services in a unified manner," in *International Conference on Service-Oriented Computing*, 2006, pp. 104–115.

[15] W. Lv and J. Yu, "pservice: Peer-to-peer based web services discovery and matching," in *International Conference on Systems and Networks Communications*, 2007, p. 54.

[16] T. Ruokolainen and L. Kutvonen, "Service typing in collaborative systems," in *International Conference on Interoperability for Enterprise Software and Applications*, 2006, pp. 343–353.

[17] D. Athanasopoulos, A. Zarras, and V. Issarny, "Service substitution revisited," in *Automated Software Engineering*, 2009, pp. 555–559.

[18] X. Dong, A. Halevy, J. Madhavan, E. Nemes, and J. Zhang, "Similarity search for web services," in *International Conference on Very Large Data Bases*, 2004.

[19] D. Athanasopoulos, A. Zarras, P. Vassiliadis, and V. Issarny, "Mining service abstractions," in *International Conference on Software Engineering*, 2011, pp. 944–947.

[20] J. Wu, L. Chen, Z. Zheng, M. R. Lyu, and Z. Wu, "Clustering web services to facilitate service discovery," *Knowledge and Information Systems*, vol. 38, no. 1, pp. 207–229, 2014.

[21] S. Dasgupta, S. Bhat, and Y. Lee, "Taxonomic clustering and query matching for efficient service discovery," in *IEEE International Conference on Web Services*, 2011, pp. 363–370.

[22] C. B. Pop, V. R. Chifu, I. Salomie, M. Dinsoreanu, T. David, and V. Acretoaie, "Semantic web service clustering for efficient discovery using an ant-based method," in *International Symposium on Intelligent Distributed Computing*, 2010, pp. 23–33.

[23] N. Zhang, J. Wang, K. He, and Z. Li, "An approach of service discovery based on service goal clustering," in *International Conference on Services Computing*, 2016, pp. 114–121.

[24] B. Cheng, S. Zhao, C. Li, and J. Chen, "MISDA: web services discovery approach based on mining interface semantics," in *International Conference on Web Services*, 2016, pp. 332–339.

[25] D. Tosi, G. Denaro, and M. Pezzè, "Towards autonomic service-oriented applications," *International Journal of Autonomic Computing*, vol. 1, no. 1, pp. 58–80, 2009.

[26] E. Riccobene and P. Scandurra, "Formal modeling self-adaptive service-oriented applications," in *Symposium on Applied Computing*, 2015, pp. 1704–1710.

[27] P. Plebani and B. Pernici, "Urbe: Web service retrieval based on similarity evaluation," *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, no. 11, 2009.

[28] V. Levenshtein, "Binary Codes Capable of Correcting Spurious Insertions and Deletions of Ones," *Problems of Information Transmission*, vol. 1, pp. 8–17, 1965.

[29] J. Munkres, "Algorithms for the assignment and transportation problems," *Journal of the Society of Industrial and Applied Mathematics*, vol. 5, no. 1, pp. 32–38, 1957.

[30] J. Tekli and R. Chbeir, "A novel xml document structure comparison framework based-on sub-tree commonalities and label semantics," *Journal of Web Semantics*, vol. 11, pp. 14–40, 2012.

[31] M. Pawlik and N. Augsten, "Efficient computation of the tree edit distance," *ACM Transactions on Database Systems*, vol. 40, no. 1, pp. 3:1–3:40, 2015.

[32] O. Maqbool and H. A. Babri, "Hierarchical clustering for software architecture recovery," *IEEE Transactions on Software Engineering*, vol. 33, no. 11, pp. 759–780, 2007.

[33] R. A. Baeza-Yates and B. A. Ribeiro-Neto, *Modern Information Retrieval*. ACM Press/Addison-Wesley, 1999.